Hybrid Security Audit

Report Version 1.0

February 5, 2024

Conducted by the **Hunter Security** team:

George Hunter, Lead Security Researcher
deadrosesxyz, Senior Security Researcher

Table of Contents

1 About Hunter Security

Hunter Security is a duo team of independent smart contract security researchers. Having conducted over 50 security reviews and reported tens of live smart contract security vulnerabilities, our team always strives to deliver top-quality security services to DeFi protocols. For security review inquiries, you can reach out to us on Telegram or Twitter at *@georgehntr*.

2 Disclaimer

Audits are a time-, resource-, and expertise-bound effort where trained experts evaluate smart contracts using a combination of automated and manual techniques to identify as many vulnerabilities as possible. Audits can reveal the presence of vulnerabilities **but cannot guarantee their absence**.

3 Risk classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	High	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

3.1 Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - involves a small loss of funds or affects a core functionality of the protocol.
- **Low** - encompasses any unexpected behavior that is non-critical.

3.2 Likelihood

- **High** - a direct attack vector; the cost is relatively low compared to the potential loss of funds.
- **Medium** - only a conditionally incentivized attack vector, with a moderate likelihood.
- **Low** - involves too many or unlikely assumptions; offers little to no incentive.

3.3 Actions required by severity level

- **High** - client **must** fix the issue.
- **Medium** - client **should** fix the issue.
- **Low** - client **could** fix the issue.

4 Executive summary

The Hunter Security team was engaged by DragonX to review the DragonX Hybrid smart contracts during the period from January 30, 2024, to February 2, 2024.

Overview

Project Name	DragonX Hybrid
Repository	https://github.com/DragonX2024888/DragonX-Hybrid
Commit hash	5a8036176dd87a227d76332865fb388ae7713850
Resolution	6cf372e20ebd1170a86783f565d5e3aaa0a0062d
Methods	Manual review

Timeline

-	January 30, 2024	Audit kick-off
v0.1	February 2, 2024	Preliminary report
v1.0	February 5, 2024	Mitigation review

Scope

contracts/DragonBurnProxy.sol
contracts/DragonHybrid.sol

Issues Found

High risk	1
Medium risk	1
Low risk	1

5 Consultants

George Hunter - a proficient and reputable independent smart contract security researcher with over 50 solo and team security engagements contributing to the security of numerous smart contract protocols in the past 2 years. Previously held roles include Lead Smart Contract Auditor at Paladin Blockchain Security and Smart Contract Engineer at Nexo. He has audited smart contracts for clients such as LayerZero, Euler, TraderJoe, Maverick, Ambire, and other leading protocols. George's extensive experience in traditional audits and meticulous attention to detail contribute to Hunter Security's reviews, ensuring comprehensive coverage and preventing vulnerabilities from slipping through.

deadrosesxyz - a proficient and reputable bug bounty hunter with over 10 live smart contract vulnerability reports, confirmed and effectively mitigated through Immunefi. He has made significant contributions to securing protocols such as Yearn, Velodrome, Euler, SPool, and other leading DeFi protocols. His creativity and experience in hunting vulnerabilities in live protocols provide unmatched value to Hunter Security's reviews, uncovering unique vulnerabilities and edge cases that most auditors overlook.

6 Findings

6.1 High

6.1.1 Ownership accounting is not done correctly on transfers

Severity: *High*

Context: DragonHybrid.sol#L64-L65

Description: The `balanceOfDragon` mapping is used to track the balance of each specific type of DragonX Hybrid NFT a user has:

```
/**
  * @notice balance of dragons per owner
  */
mapping(address owner => mapping(DragonTypes dragonType => uint256 balanceOf))
    public balanceOfDragon;
```

It is incremented by 1 upon mint and decremented by 1 accordingly upon burn.

The problem is that the mapping is not updated on ERC721 transfers which would cause the `burn` method to revert making it impossible for new owners to unlock the deposited DragonX tokens.

Recommendation: Consider properly tracking the `balanceOfDragon` of users when ERC721 transfers are executed.

Resolution: Resolved. The recommended fix was implemented.

6.2 Medium

6.2.1 The locked amount for an NFT is returned to the caller instead of the owner

Severity: *Medium*

Context: DragonHybrid.sol#L252-L254, DragonHybrid.sol#L268-L269

Description: When a user mints a DragonX Hybrid NFT, they lock an amount of DragonX tokens that are released when the NFT is burned:

```
// Release tokens to NFT owner
dragonX.safeTransfer(_msgSender(), lockAmount);
```

The intended behavior as stated in the above comment is to return the funds back to the owner of the NFT. However, the tokens are returned to the `_msgSender()` which is not necessarily the owner. The authentication in the `DragonHybrid.burn` is done in the following way:

```
// Setting an "auth" arguments enables the ‘_isAuthorized‘ check which verifies that
    the token exists
// (from != 0). Therefore, it is not needed to verify that the return value is not 0
    here.
address from = _update(address(0), tokenId, _msgSender());
```

The `_update` implements a check that the passed address (the third parameter) is either the owner *or an approved address* of this `tokenId`.

Therefore, the transfer recipient should be the `from` address variable instead of the `_msgSender()`.

Recommendation: Consider whether the intended behavior is to allow approved addresses to burn owner's tokens. If that's not the case, implement the following change:

```
// Release tokens to NFT owner
dragonX.safeTransfer(from, lockAmount);
```

Resolution: Resolved. The recommended fix was implemented.

6.3 Low

6.3.1 Default value of enum may return misleading information

Severity: *Low*

Context: Constants.sol#L13-L25

Description: The enum `DragonTypes` includes the following elements:

```
// Dragon Types
enum DragonTypes {
    // 8 Million DragonX lock up, mint fee: 800 K TitanX, burn fee: 80 K DragonX
    Apprentice,
    // 88 Million DragonX lock up, mint fee: 8 Million TitanX, burn fee: 800 K
        DragonX
    Ninja,
    // 888 Million DragonX, mint fee: 88 Million TitanX, burn fee: 8 Million DragonX
    Samurai,
    // 8 Billion DragonX, mint fee: 888 Million TitanX, burn fee: 88 Million DragonX
    Shogun,
    // 88 Billion DragonX, mint fee: 8 Billion TitanX, burn fee: 888 Million DragonX
    Emperor
}
```

It is used to determine the rank of a DragonX Hybrid NFT by storing it in the `tokenIdToDragonType` mapping upon mint:

```
/**
  * @notice maps NFT id to its metadata
  */
mapping(uint256 => DragonTypes) public tokenIdToDragonType;

// ...

function mint(
    DragonTypes dragonType
) external returns (uint256 newTokenId) {
    // ...

    tokenIdToDragonType[newTokenId] = dragonType;

    // ...
}
```

The problem is that the default value of the enum is a valid element, which means that when an invalid `tokenId` is passed to the `tokenIdToDragonType` mapping, a valid dragon type will be returned. This case is handled in the rest of the functions of the contract, but an external contract or an off-chain program may retrieve wrong value if they rely on the value read using the `tokenIdToDragonType` mapping's default getter function.

Recommendation: Consider making the `tokenIdToDragonType` private and only return the `tokenId`'s `dragonType` in a custom getter function if the `tokenId` exists (has an owner);

Resolution: Resolved. The recommended fix was implemented.